

AD-A130 343

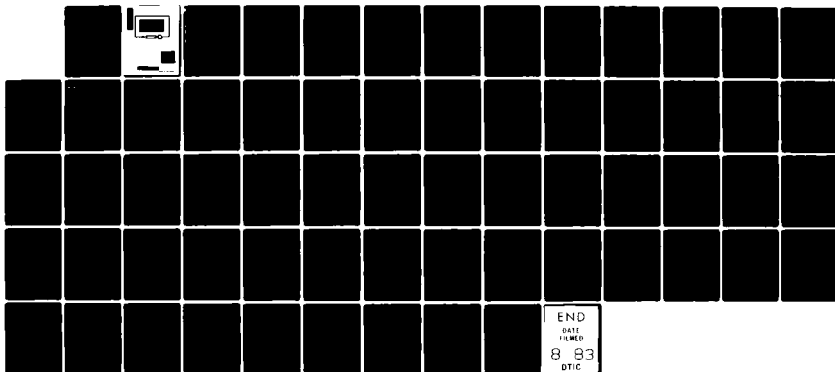
ON THE FEASIBILITY OF INTERFACING POD (PERFORMANCE
ORIENTED DESIGN) AND P..(U) BGS SYSTEMS INC WALTHAM MA
J P BUZEN ET AL. DEC 80 TR-80-018 N00039-79-C-0468

1/1

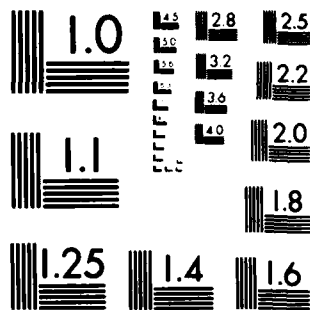
UNCLASSIFIED

F/G 9/2

NL



END
DATE
FILMED
8 83
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

On the Feasibility of Interfacing
POD and PSL/PSA

Contract N00039-79-C-0468



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A130 343	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) On the Feasibility of Interfacing POD and PSL/PSA		5. TYPE OF REPORT & PERIOD COVERED
5. AUTHOR(s) J.P. Buzen, R.P. Goldberg, D.E. Hall, K. Hua, A.M. Langer, A.I. Levy, P.S. Mager, H.S. Schwenk		6. PERFORMING ORG. REPORT NUMBER TR-80-018
7. PERFORMING ORGANIZATION NAME AND ADDRESS BGS Systems, Inc. WALTHAM MA		8. CONTRACT OR GRANT NUMBER(s) N00039-79-C-0468
9. CONTROLLING OFFICE NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE December, 1980
13. DISTRIBUTION STATEMENT (of this Report)		14. SECURITY CLASS. (of this report) Unclassified
15. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		16. DECLASSIFICATION/DOWNGRADING SCHEDULE
17. SUPPLEMENTARY NOTES		
18. KEY WORDS (Continue on reverse side if necessary and identify by block number) POD, PSL/PSA, Performance, Modeling, Program Specification Language, Program Design, Performance Oriented Design Requirements Specification		
19. ABSTRACT (Continue on reverse side if necessary and identify by block number) This report addresses the feasibility of and requirements for interfacing two powerful software engineering tools - POD (Performance Oriented Design) developed by BGS Systems, Inc. and PSL/PSA (Problem Statement Language/Problem Statement Analyzer), developed by the ISDOS project at the University of Michigan. POD is a software engineering tool for the life cycle management of system performance. It is a tool which provides a structured machine readable format for representing a system's hardware architecture.		

This document has been approved
for public release and sale; its
distribution is unlimited.

DD FORM 1 JAN 79 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

software structure and external load demand and, based on this representation, provides the vehical to calculate system performance values such as response time, throughput and device utilization. PSL/PSA is a computer-aided structured documentation and analysis technique used for analysis and documentation of requirements and preparation of functional specifications. The capability to describe systems in computer processible form is based on the system description language PSL. The ability to record the description in a data base, incrementally modify it, and perform analysis is based on the problem statement analyzer PSA.

In considering the feasibility of and requirements for interfacing POD with PSL/PSA, PSL/PSA is assessed in terms of its ability to describe the required POD input specifications. In addition consideration is given to possible PSL/PSA extensions to support POD and an approach for interfacing these tools is determined.

Mission For	
GRA&I	<input checked="checked" type="checkbox"/>
TAB	<input type="checkbox"/>
Advanced	<input type="checkbox"/>
<i>Little on file</i>	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
<i>A</i>	



UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ON THE FEASIBILITY OF INTERFACING
POD AND PSL/PSA

30 December 1980

BGS Systems, Inc.
Box 128
Lincoln, MA 01773

Table of Contents

1 INTRODUCTION	5
2 OVERVIEW OF POD AND PSL/PSA	7
2.1 POD	7
2.2 PSL/PSA	11
3 COMPARISON OF APPROACHES	15
3.1 Input Specifications	15
3.1.1 Software Structure	16
3.1.2 Configuration Environment	21
3.1.3 Resource Consumption	22
3.1.4 Load Demand	24
3.1.5 Other Aspects	26
3.2 Reporting Facilities	27
3.2.1 Common Facilities	27
3.2.2 POD Specific Facilities	32
3.2.3 PSL/PSA Specific Facilities	34
3.3 Functional Processing	36
3.3.1 PSL/PSA Approach	36
3.3.2 POD Approach	37
4 INTERFACE FEASIBILITY AND REQUIREMENTS	41
4.1 Direct Translation	41
4.1.1 Configuration Specification	43
4.1.2 Workload Specification	44
4.1.3 Software Specification	46
4.1.4 Defining System Parameters	49
4.2 Potential Extensions to PSL to Facilitate Performance Specification	50
4.3 Interactive Tools to Aid the Translation Process	51
5 SUMMARY AND CONCLUSIONS	57

List of Figures

Figure 2-1: Overview of POD	8
Figure 2-2: Overview of PSL/PSA	11
Figure 3-1: Network Structures	16
Figure 3-2: Tree Structure	17
Figure 3-3: Utilizes Network Structure	17
Figure 3-4: PSC Dynamic Interaction Matrix	31
Figure 3-5: Sample FTS Interaction	34
Figure 3-6: SDL/SDM Simulation Overview	37
Figure 3-7: Basic POD/PC Model	39

EXECUTIVE SUMMARY

This report addresses the feasibility of and the requirements for interfacing two powerful software engineering tools - POD (Performance Oriented Design) developed by BGS Systems and PSL/PSA (Problem Statement Language/Problem Statement Analyzer) developed by the ISDOS project, University of Michigan. POD is a software engineering tool designed to evaluate the performance of systems based on specifications of their design. PSL/PSA is designed for documentation and analysis of functional and requirements specifications.

In considering the feasibility of and requirements for an interface between these tools, PSL/PSA was assessed in terms of its ability to describe required POD input specifications. It was found that a significant portion of the required POD input currently exists in PSL/PSA but that a small number of performance parameters would need to be incorporated into any PSL/PSA translator (manual or automated) for a complete POD input specification. Such information is required because of the significantly differing scopes of the two tools and the level of information required by each to support their respective objectives. For this same reason, however, it is believed that creation of a POD-PSL/PSA interface would provide a more complete and comprehensive software engineering environment than is provided currently by either tool alone.

FOREWORD

Development of the POD System by BGS Systems, Inc. has been supported in part by the following contracts awarded by the Department of the Navy: N00039-77-C-0240, N00039-79-C-0458, and N00039-79-C-0468.

Principal members of BGS personnel participating in the effort include Dr. J.P. Buzen, Dr. R.P. Goldberg, Mr. D.E. Hall, Mr. K. Hua, Mr. A.M. Langer, Mr. A.I. Levy, Dr. M.H. Liu, Mr. P.S. Mager, and Dr. H.S. Schwenk, Jr. In addition, Dr. D Teichroew of the University of Michigan, provided key technical PSL/PSA information.

Technical guidance and supervision were provided by Mr. John Machado of NAVELEX and Mr. Linwood Sutton of NOSC.

DISCLAIMER

This document represents the completion of the first phase in the development of a POD - PSL/PSA interface. Its purpose is to examine the feasibility of and requirements for such an interface. It is important to note, however, that both PSL/PSA and POD are in a continuing process of development. Therefore, specifications in this document should not be construed as being final. Rather the objective is to provide a working document to serve as the focus for future discussions as the tools develop and the interface requirements become clear.

1 INTRODUCTION

This report addresses the feasibility of and requirements for interfacing two powerful software engineering tools - POD (Performance Oriented Design) developed by BGS Systems Inc. and PSL/PSA (Problem Statement Language/Problem Statement Analyzer) developed by the ISDOS project at the University of Michigan. POD is a software engineering tool for the life cycle management of system performance. It is a tool which provides a structured machine readable format for representing a system's hardware architecture, software structure and external load demand and, based on this representation, provides the vehicle to calculate system performance values such as response time, throughput and device utilization. PSL/PSA is a computer-aided structured documentation and analysis technique used for analysis and documentation of requirements and preparation of functional specifications. The capability to describe systems in computer processible form is based on the system description language PSL. The ability to record the description in a data base, incrementally modify it, and perform analysis is based on the problem statement analyzer PSA.

In considering the feasibility of and requirements for interfacing POD and PSL/PSA, PSL/PSA is assessed in terms of its ability to describe the required POD input specifications. In addition, consideration is given to possible PSL/PSA extensions to support POD and an approach for interfacing these tools is determined.

The report is organized as follows. Chapter 2 provides a brief overview of each tool. Chapter 3 then compares the similarities and differences in each approach. Chapter 4 addresses the feasibility and interface approaches for the tools. And finally, a summary of the major findings is presented in Chapter 5.

2 OVERVIEW OF POD AND PSL/PSA

In this chapter a brief overview of POD and PSL/PSA is presented. The overviews are not intended to be comprehensive, rather the purpose is to present an introductory description of the facilities in each tool. More specifically, the overview serves to point out the major objectives of each tool - performance analysis for POD and functional requirements specification for PSL/PSA.

2.1 POD

An overview of the POD system is presented in Figure 2-1. As indicated in this figure, POD consists of a System Description Subsystem, Model Generation and Analysis Subsystem, and Interactive Evaluation Subsystem. The System Description Subsystem accepts and interprets a high level language description of the system or proposed system design. The Model Generation and Analysis Subsystem automatically generates an internal service demand model of the system, in the form of a queueing behavior network, and evaluates the steady state queueing behavior of the model. The Interactive Evaluation Subsystem provides for on-line control of the former two components. It provides a user interface for varying model and system description parameters, performing parametric analysis of the changes, and generating a variety of performance related reports.

We discuss each of these subsystem in more detail below.

POD System Description Subsystem

The POD System Description Subsystem provides an analyst with the capability of describing the system and environment in a highly structured, machine processable form. This facility permits the specification of each hardware and software component, their interactions and their interconnections.

The POD System Description Subsystem accepts an input file that consists of three fundamental components - the Configuration Specification, the Workload Specification and the Module Specification Sections. The Configuration Specification Section allows the analyst to describe a system's hardware configuration. Included are language constructs for describing the gross configuration (e.g., number of processors, data paths, etc.) and characteristics of individual hardware devices (e.g., device processing speed, storage capacity, etc.).

The Workload Specification Section serves to capture the characteristics of the system's external environment. The Workload Specification Section provides the facility to describe a system's external work demand or workload(s). In this context, the term workload refers to a stream of jobs arriving at the system from some external source. In the Workload Specification Section designers are provided with language constructs for specifying such information as the arrival rate of individual requests for processing, relative priority and so on.

Pertinent characteristics of an individual workload task are specified in the Module Specification Section. The Module Specification Section provides a facility for the description of module characteristics, independent of the specific task that may be invoking the module. This includes logical flow of control and resource consumption information.

A complete system description, consisting of a Configuration Specification Section, a Workload Specification Section, and a Module Specification Section, is known as the POD System Description File. Through the POD System Description Subsystem the analyst initially prepares a System Description File that represents a gross and largely simplified description of his system. The file is then successively refined as design and implementation decisions become firm. As each stage of the design is completed, the System Description File reflects a successively more detailed and accurate description of the overall system.

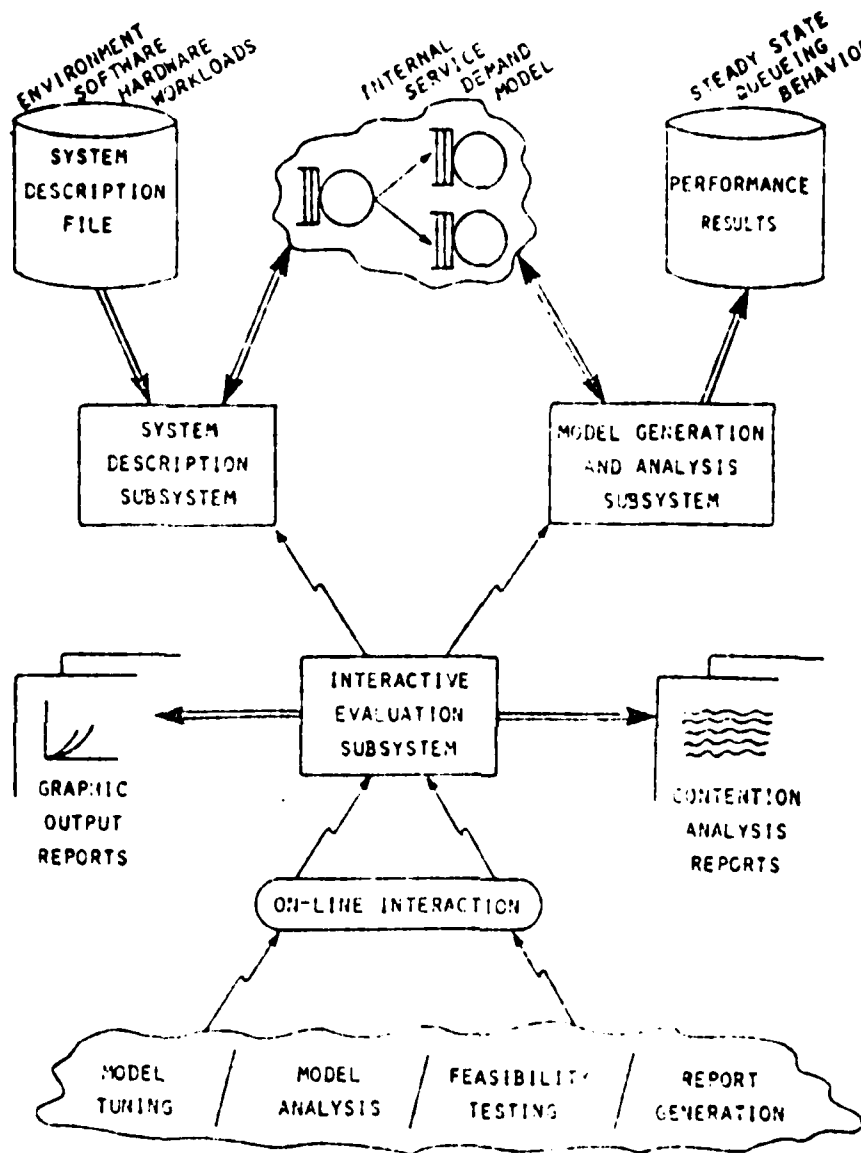


Figure 2-1: Overview of POD

BGS SYSTEMS, INC.



One University Office Park • Waltham, MA 02254 • Tel. 617-891-0000 • Telex 923314

March 10, 1983

Defense Technical Information Center
Bldg 5
Cameron Station
Alexandria, VA 22314

Dear Sirs:

Here are the first three (3) reports we are releasing under the DTIC program. Feel free to contact me at (617) 891-0000 if you have any questions or have suggestions as to the best way for us to format and/or package future reports we release under your program.

These 3 reports were produced under a Navy research contract from NAVELEX. Please contact R. Fratila at ELEX 613 (telephone: (202) 692-6089) for distribution classification.

Very truly yours,

Peter Mager
BGS SYSTEMS, INC.

PM:cwc

Enc. 3 copies of: On the Feasibility of Interfacing
POD and PSL/PSA

Possible Approaches for Interfacing
POD to PSL and SEM

POD-HOS Interfaces: An Examination of
Axes/POD Relationships and Other Issues

Model Generation and Analysis Subsystem

The POD Model Generation and Analysis Subsystem consists of a Model Generation Facility and a Model Analysis Facility. The Model Generation Facility accepts and interprets the POD System Description File. This facility automatically constructs a performance model of the input formulated as a queueing network. Such models are an extension to those found in classical queueing theory, and are sufficiently rich to portray the characteristics of a complex computer system to a very high degree of accuracy.

The Model Analysis Facility accepts the performance models generated by the Model Generation Facility and produces, under user control, a variety of performance prediction reports. These reports provide summary information on the expected performance of the design. Moreover, many of these reports have been specifically designed to focus into areas of system congestion and clearly identify the causes of poor performance. The method of solution is based on computational queueing network algorithms supplemented by approximation techniques. These algorithms provide solutions to models orders of magnitude faster than conventional simulation techniques and allow for an on-line interactive mode of operation.

Interactive Evaluation Subsystem

The third major software component of POD is the Interactive Evaluation Subsystem. This subsystem provides POD with a user oriented front end for interactive evaluation of design/implementation alternatives. It permits the on-line user to vary aspects of the hardware and/or software specifications and interactively determine the performance impact of the alternatives.

The Interactive Evaluation Subsystem is designed to permit parametric performance studies to be easily performed. The user can, in a single high level command, specify the design parameters to be altered and the performance objectives to be achieved. In this case the Model Generation

and Analysis Subsystem is iteratively invoked, analyzing models of the design alternatives at each iteration, until the performance objectives are satisfied.

The Interactive Evaluation Subsystem also provides the user with a performance report generation capability. Facilities for creating specially tailored reports are available in addition to a wide variety of predefined performance summary reports. Further, an interface to a graphics facility for conversion of results to graphic form is provided.

2.2 PSL/PSA

PSL is a language for describing systems¹. PSL is intended to be used in situations in which analysts now describe systems. The descriptions of systems produced using PSL are used for the same purposes as that produced manually. PSL may be used both in batch and interactive environments, and therefore only "basic" information about the system need to be stated in PSL. All "derived" information can be produced in hard copy form as required.

The Problem Statement Language is based first on a model of a general system, and secondly on the specialization of the model to a particular class of systems, namely information systems. The model of a general system is relatively simple. It merely states that a system consists of things which are called OBJECTS. These objects may have PRIORITIES and each of these PROPERTIES may have PROPERTY VALUES. The objects may be connected or interrelated in various ways. These connections are called RELATIONSHIPS.

The objective of PSL is to be able to express in a syntactically analyzable form as much of the information which commonly appears in System

¹The material for this section has been obtained, for the most part, directly from PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems, D. Teichrow and E.A. Hershey III, IEEE Transactions on Software Engineering, January 1977.

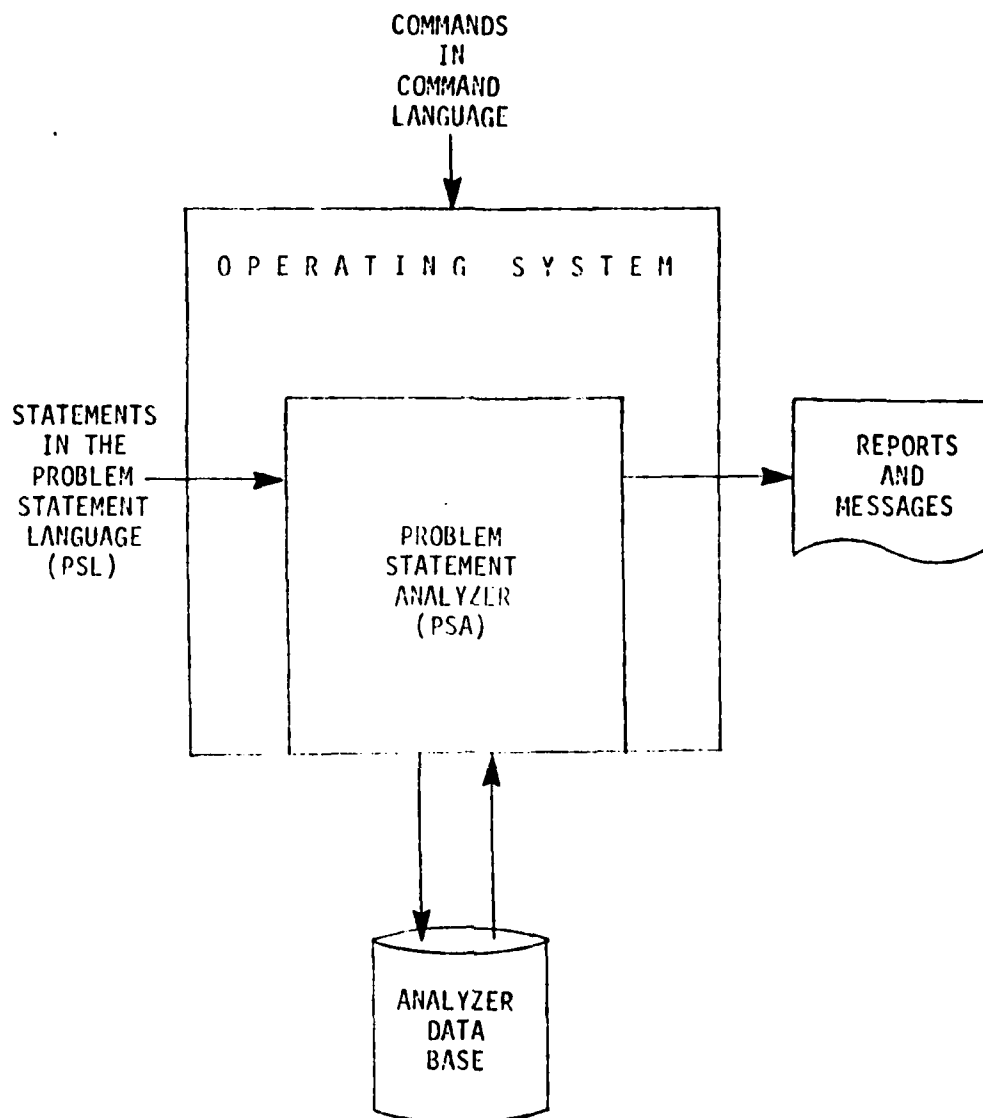


Figure 2-2: Overview of PSL/PSA

Definition Reports as possible. System Descriptions may be divided into the following major aspects:

- The System Input/Output Flow aspect of the system deals with the interaction between the target system and its environment.
- System Structure is concerned with the hierarchies among objects in a system. System structures can represent high-level hierarchies which may not actually exist in the system, as well as those that do.
- The Data Structure aspect of system description includes all the relationships which exist among data used and/or manipulated by the system as seen by the "users" of the system.
- The Data Derivation aspect of the system description specifies which data objects are involved in particular PROCESSES in the system. It is concerned with what information is used, updated, and/or derived, how this is done, and by which processes.
- The System Size and Volume aspect is concerned with the size of the system and those factors which influence the volume of processing which will be required.
- The System Dynamics aspect of system description presents the manner in which the target system "behaves" over time. All objects (of a particular type) used to describe the target system have characteristics which distinguish them from other objects of the same type. Therefore, the PROPERTIES of particular objects in the system must be described. The PROPERTIES themselves are objects and given unique names.
- The Project Management aspect requires that, in addition to the description of the target system being designed, documentation of the project designing (or documenting) the target system be given. This involves identification of people involved and their responsibilities, schedules, etc.

As information about a particular system is obtained, it is expressed in PSL and entered into a data base using the Problem Statement Analyzer (PSA). At any time standard outputs or reports may be produced on request. The various reports (which are presented in detail in the following chapter) can be classified on the basis of the purposes which they serve.

The major benefits claimed for computer-aided documentation such as provided by PSL/PSA are that the "quality" of the documentation is improved and that the cost of design, implementation, and maintenance will be reduced. The "quality" of the documentation, measured in terms of

preciseness, consistency, and completeness is increased because the analysts must be more precise. The software performs checking, and the output reports can be reviewed for remaining ambiguities, inconsistencies, and omissions. While completeness can never be fully guaranteed, one important feature of the PSL/PSA computer-aided method is that all the documentation that "exists" is in the data base, and therefore the gaps and omissions are more obvious. Consequently, the organization knows what data it has, and does not have to depend on individuals who may not be available when a specific item of data about a system is needed. Any analysis performed and reports produced are up-to-date as of the time it is performed. The coordination among analysts is greatly simplified since each can work in his/her own area and still have the system specifications be consistent.

Development will take less time and cost less because errors, which usually are not discovered until programming or testing, have been minimized. It is recognized that one reason for the high cost of systems development is the fact that errors, inconsistencies, and omissions in specifications are frequently not detected until later stages of development: in design, programming, systems tests, or even operation. The use of PSL/PSA during the specification stage reduces the number of errors which will have to be corrected later. Maintenance costs are considerably reduced because the effect of a proposed change can easily be isolated, thereby reducing the probability that one correction will cause other errors.

3 COMPARISON OF APPROACHES

As described in the previous chapter, both POD and PSL/PSA are attempting to reduce life cycle costs of system development by providing the system development community with automated software engineering tools. POD has as its focus the performance analysis of design alternatives. PSL/PSA has as its focus functional and requirements specifications.

There is, out of necessity, some overlap in the functionality of these tools. This overlap exists, for the most part, in the area of structured system documentation. This is because in order to do either a functional/requirements analysis (PSL/PSA) or a design/performance analysis (POD) detailed specifications of system environment, flow and operation need be created. There are also areas where the system documentation provided by each tool differs significantly. This is because the level of information required in the specification for functional/requirements purposes differs from that required for performance analysis purposes. It is the purpose of this chapter to compare and contrast the similarities and differences in the approaches used by POD and PSL/PSA.

3.1 Input Specifications

The POD System Description Subsystem accepts an input file that consists of three fundamental components - the Module Specification, the Configuration Specification, and the Workload Specification Sections. A complete system description, consisting of the above three sections, is referred to as the POD System Description File. This file represents a largely simplified description of the target system, but one that is sufficiently detailed to portray its performance requirements.

The Problem Statement Language (PSL) is a language for describing the function and requirements of systems. It is based on a model of a general system and the specialization of the model to a particular class of systems (i.e. information systems). A system in this model consists of "objects". These objects may have "properties" with property values. And the

connections among these objects are termed "relationships". An information system is modeled by a limited number of predefined objects, properties, and relationships.

Below we explore the major aspects of the input specifications for each of PSL/PSA and POD.

3.1.1 Software Structure

The definition of a software structure is usually introduced to facilitate particular software engineering approaches such as "structured programming" and "top-down design". This structure is based on pertinent control flow information. The building blocks of this structure are often termed "modules", "subroutines", and so on. Each of POD and PSL/PSA have input specifications pertaining to the software structure.

In POD, the term "MODULE" is adopted. The interrelationship concerning control flow among modules is represented by the CALL statement. For example, if the statement CALL BETA is included in association with the specification of module ALPHA, the relationship "ALPHA calls BETA" is established in the structure. A module can be called by several other modules in POD and, conversely, several modules can call the same module.

If a graphical diagram is used to illustrate the software structure in POD, each module can be represented by a node and each call statement by a directed arc pointing from the calling module to the called module. This graph may appear to be an acyclic directed network. In other words, modules and call statements in POD constitute a "network structure" as depicted in Figure 3-1.

In contrast, PSL allows a wide range of objects to be structured by using the "SUBPARTS ARE" and "PART OF" statements. Both information structures and processing structures can be set up. However, if the structure defined by the "subparts are" and "part of" interrelationships is depicted in graphical form, it will be a directed tree rather than a network. This is because of the constraint in PSL that each object cannot be part of several distinct objects and several objects cannot have the same subparts.

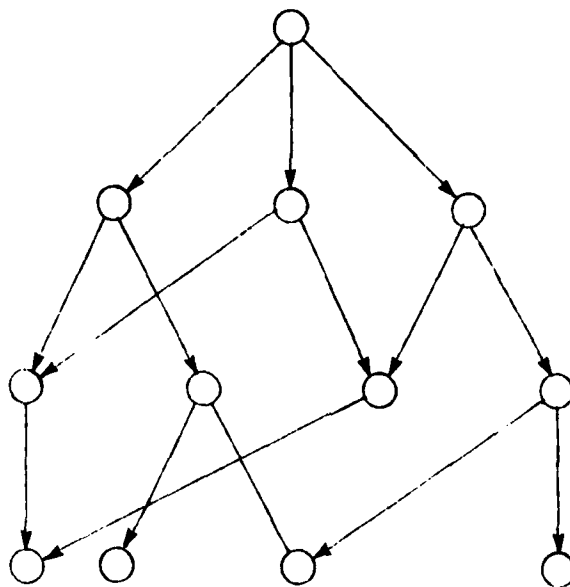


Figure 3-1: Network Structures

The tree structure established by the "subparts are" and "part of" relationships is shown in Figure 3-2.

PSL also allows a structure to be constructed on the basis of "UTILIZES" and "UTILIZED BY" statements, in the PROCESS section. A process may utilize other processes and it may be utilized by other processes. Notice that several different processes can utilize the same process and a process can be utilized by several different processes. Therefore, this structure is a network structure as we discussed previously. For example, the following PSL statements define a network structure as shown in Figure 3-3.

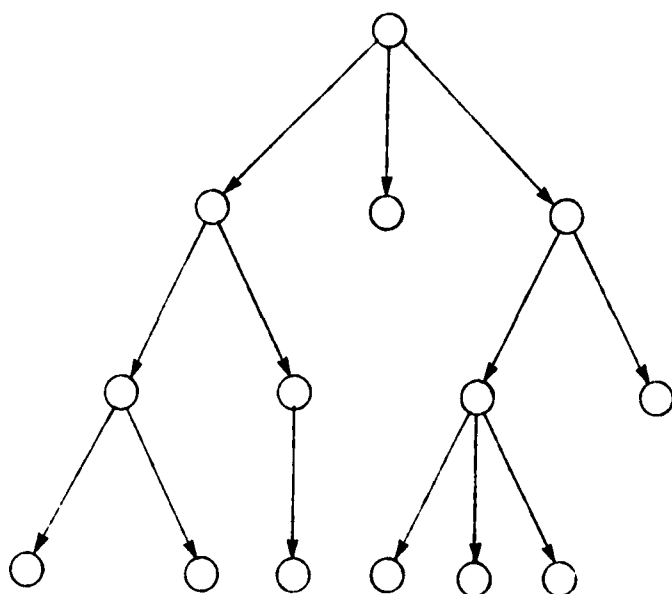


Figure 3-2: Tree Structure

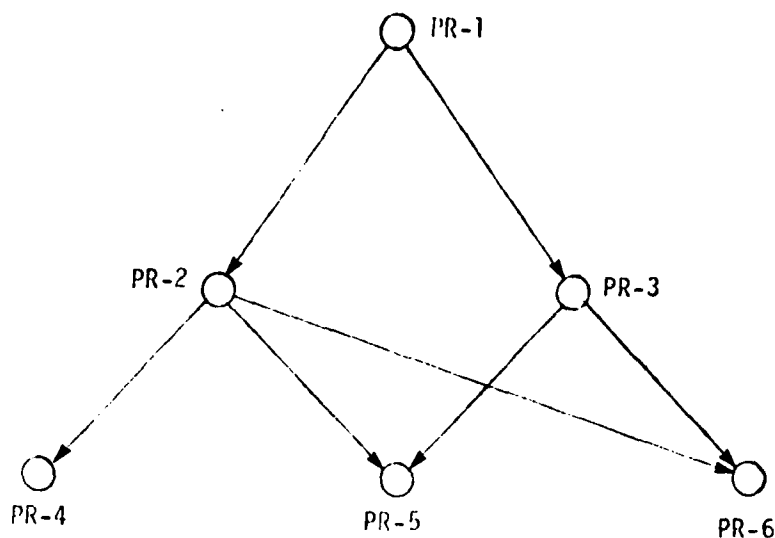


Figure 3-3: Utilizes Network Structure

DEFINE	PROCESS	PR_1;
	UTILIZES	PR_2; PR_3;
DEFINE	PROCESS	PR_2;
	UTILIZES	PR_4, PR_5, PR_6;
DEFINE	PROCESS	PR_3;
	UTILIZES	PR_5, PR_6;

Closely related to the concept of software structure is algorithmic flow of control. This refers to specific primitives embedded in the software specification language which are capable of describing complex algorithms. The PSL approach to this issue, for the most part, is through comments embedded in the specification. This is sufficient for PSL purposes since the PSL objective is functional and requirements specification. In contrast, such algorithmic issues are an integral part

of the POD software specification. This is because POD addresses performance design at a level that includes algorithmic functioning.

Examples of algorithmic constructs embedded in the POD software specifications are the LOOP and TEST constructs. The LOOP construct specifies the repeated execution of a block of module statements. The TEST construct specifies conditional branching within a module. Together these provide the facility for a refined description of each module's resource consumption behavior at a level of detail that may be present in complex algorithms. For example, a module may call one of several other modules depending upon testing of a switch:

```
TEST FILE
CASE 'CLAIM'
    CALL CLAIM_PROCESSING
CASE 'PEND'
    CALL END_PROCESSING
CASE 'POLICY'
    CALL POLICY_PROCESSING
ENDTEST
```

Or, a block of module statements may be executed repeatedly for as many times as specified by a performance parameter or a complex function based on various performance parameters. For example:

```
LOOP    NO OF REQUESTS  TIMES
        EST LOCAL DELTA USAGE = 6 READ
        CALL RESPONSE
ENDLOOP
```

Moreover, to aid in the mapping of algorithms to POD system specification syntax, an optional list of module parameters may be specified and passed to the module being called. This facility provides a convenient way to represent the control flow information and is commonplace in high-level algorithmic programming languages.

5.1.2 Configuration Environment

An important part of a system's specification for performance analysis purposes and also for functional and requirements analysis is the specification of a system's hardware environment. This is an integral part of the POD input file. In POD, the Configuration Specification Section provides facilities for specifying individual hardware devices (e.g. processors, channels, disks, buses) and, in the case of I/O devices, pertinent information concerning the files resident on the devices. The ATTACH attribute is used in conjunction with CPU, CHANNEL, and CONTROLLER device types to represent the data paths within the configuration. This permits specification of a data path from CPU to CHANNEL, CHANNEL to CONTROLLER, and CONTROLLER to I/O devices/ The specification may be important because of the fact that I/O timing consists of I/O device timing as well as data path timing.

In POD information is also supplied on the characteristics of devices in the configuration. For example, MIP rate (or millions of instructions per second) is supplied to specify processor capacity. Other examples include the SEEK, REVOLUTION, and RATE primitives for specifying the seek time, revolution time and transfer rate of I/O devices. In addition, extensions are currently underway to allow specification of specialized operations on tactical devices. An example of this is slew rate for a gun firing device.

A PSL description contains two sections that can provide information about the hardware environment in which a computerized application is to be run: the PROCESSOR section and the RESOURCE section. These describe what in POD would be called devices and device types. A PROCESSOR is linked to a given RESOURCE by a CONSUMES statement in the PROCESSOR section or a CONSUMED BY statement in the RESOURCE section. There can be one or more ATTRIBUTES associated with each resource. These ATTRIBUTES can be used to specify the equivalent of operations in POD. A numerical value, character string or system parameter can be associated with each attribute. This assignment can take place in either the PROCESSOR or RESOURCE section. The PROCESSOR in turn can be associated with the software modules it runs by a PERFORMS or PERFORMED BY statement.

Topology could be specified in PSL by defining an attribute called "attached-to" in the PROCESSOR section. This attribute could then be assigned as value a list of all devices that are directly attached to that processor. It should be noted however, the list would be treated simply as an uninterpreted character string by PSL/PSA.

3.1.3 Resource Consumption

The specification of resource consumption information, which is addressed by both POD and PSL, is essential to performance calculation function. The term resource may refer to the processing power of the computer system such as the CPU and I/O device times. The execution of a job stream consumes these resources.

POD provides the facility for specifying resource consumption information in the body of a module which is the basic execution unit of a job stream. This information is specified by naming the device and the activity performed on that device by the module. In the case of I/O devices, the resource consumption information may be optionally specified by naming specific files and the activity on those files. POD will automatically calculate the specific hardware resources consumed as a result of the file activity.

For example, if the estimated CPU execution time of a module is 1960 milliseconds from entry to exit including all the sub-modules invoked, the statement EST TOTAL CPU USAGE = 1960 MSEC can be used to represent this situation. Similarly, if the module is estimated to request 7 read operations from the file DELTA from entry to exit, the following statement may be used: EST TOTAL DELTA USAGE = 7 READS. Note that in this example the MSEC and READ are unit keywords which measure the associated resource consumption values. Where MSEC is a physical unit, READ is a logical unit. The physical resource consumed due to each read operation depends upon the characteristics of the hardware device in which the file DELTA resides and the manner in which the file is accessed. Similarly, CPU resource consumption can be specified in a logical manner by describing the number of instructions a module consumes. In this case the power of the processor or MIP rate (millions of instructions per second) is used to resolve to

total CPU resources consumed.

In PSL, a processor is the basic object that actually consumes resources and performs processes. The CONSUMES statement in the processor section specifies the name and the amount of resources that are consumed by the processor. An object type called UNIT is used for describing the unit in which resource consumption values are measured. For example:

```
DEFINE PROCESSOR CPU;
CONSUMES CPU-time AT RATE OF CPU-usage
PER no-of-instructions;
```

Here, CPU-time is a resource, CPU-usage is a system-parameter, and no-of-instructions is an attribute. The value of this attribute may be specified within the definition body of a process performed by the processor CPU. Different processes may have different attribute values for the same attribute no-of-instructions. The amount of resource consumed by the processor for the execution of a process may be computed by multiplying the value of the RATE in the processor CONSUMES statement by the value of the attribute defined for the process. For example, assume that the value of the system-parameter CPU-usage is defined to 0.01 and that the value of the attribute no-of-instructions is defined to be 3000 for process ALPHA and 4600 for process BETA; then the processor CPU-1 consumes 30 (3000×0.01) units of resource CPU-time whenever it performs ALPHA and 46 (4600×0.01) units whenever it performs process BETA. The unit in which the resource consumption values are measured in PSL is defined in the resource section.

It is also possible to associate more than one attribute for a process in PSL. Therefore the resource consumption information can be specified independently for several resources. For example, the attributes no-of-instructions, main-memory-used, and no-of-file-delta-reads may be defined for each individual process to reflect the consumption of various resources.

It is often desired that the resource consumption information be represented in a symbolic machine-independent form (e.g. number of central processor instructions, number of read/write operations). This independency allows the design to be "run" on various actual and proposed hardware, and the results compared. In this case, a facility to map

specific times on a given device to the symbolic form is required. The Configuration Specification in POD provides this facility. Among other things, it provides the facility to describe the characteristics of individual devices (such as MIP RATE for a CPU, and TRANSFER RATE, SEEK, REVOLUTION_TIME for a disk device) and pertinent information about the files resident on the I/O devices (such as BLOCK_FACTOR, RECORD_SIZE, and FILE_SIZE).

For example, the CPU time consumed by a module may be devised from the product of the number of CPU interactions executed within this module and the execution time of each individual CPU instruction (assuming that the execution time is the same for all instructions). The existence of this product form is significant in achieving machine independence for describing the target system. It allows for the possibility to specify CPU usage in the module definition body and CPU MIP RATE in the CPU device definition body. If the CPU is replaced by a faster CPU with higher MIP RATE, the hardware characteristics is updated but the software structure and the module specification is left unaffected.

PSL also provides the facility to represent resource consumption information in a machine-independent form. Performance-oriented attributes such as no-of-instructions, no-of-file-reads, no-of-file-writes may be specified for each process. The hardware characteristics are represented by RATE in the CONSUMES statement for each processor or the CONSUMED BY statement for each resource. However, the RATE is specified by a system-parameter which has a global scope. A different system-parameter specifying the resource consumption rate is defined for each different device attribute or hardware characteristic.

3.1.4 Load Demand

The performance of a computer system is not only dependent upon the system's processing capacity per se but is also dependent upon the amount of processing requested. Generally, the term workload is used to refer to this concept. A workload represents a stream of jobs (requests for processing) arriving at the computer system from some external source and is the major entity on which all performance results should be based.

The POD Workload Specification Section provides facilities for specifying the workload characteristics. Five workload types are defined in POD: INTERACTIVE, TRANSACTION, PERIODIC, CYCLE and ONE-SHOT. An INTERACTIVE workload is generated from a set of terminals. When a job completes, a response is sent to the terminal and a new job is generated. A TRANSACTION workload specifies a load external to the system, and the job arrival rate is independent of the amount of time it takes to process an individual job. A PERIODIC workload specifies that a single job enters the system in regular periodic intervals. A CYCLE workload describes the job stream that is driven by a constant backlog. That is, a statistically identical job is initiated as soon as a previous one terminates. A ONE-SHOT workload specifies a single procedural operation.

The characteristics or attributes of the workload being described depend on the workload type. The following two attributes may be specified for all workload types: JOB_STREAM and PRIORITY. JOB_STREAM is used to specify the stream of jobs or processing requests which make up the workload. PRIORITY may be used to describe a workload's relative processor priority as compared to all other workloads.

In addition, the MPL (multiprogramming level) refers to the level of processing overlap in a workload's job stream. This is required for each of the CYCLE, TRANSACTION, and INTERACTIVE workload types. The ARRIVAL RATE attribute is required for the PERIODIC and TRANSACTION workload types to specify the job arrival rate. The INTERACTIVE workload requires the USERS attribute to specify the number of users actively interacting with the system via the terminals and the THINK_TIME attribute to specify the average time between the system's response to a given interaction and the user's initiation of the next interaction.

The concept of system size and volume in PSL is similar to that of workload. The workload is specified via the HAPPENS statement in INPUTS, OUTPUTS, PROCESSES, and EVENT sections. For example:

```
DEFINE    PROCESS    DB UPDATE;
HAPPENS   600        TIMES-PER hour,
```

The above description indicates that this process will occur 600 times for each hour. Therefore the resource consumption caused by this process in a

one-hour interval is 600 times the resource consumption for the execution of this process each time.

If the object type EVENT is adopted to represent the workload, the HAPPENS statement is similar to the ARRIVAL_RATE attribute, and the job stream can be "triggered" by the event. For example:

```
DEFINE  EVENT  DB PROCESSING;
HAPPENS 10 TIMES-PER minute;
TRIGGERS DB_UPDATE;
```

3.1.5 Other Aspects

In the previous subsections we examined those parts of POD and PSL/PSA input specifications which pertain mostly to the basic input requirements for performance analysis. As noted above, some of the required information is specified, by both POD and PSL. However, PSL has as its orientation functional and requirement specification and therefore has available this type of input specification where as POD does not. This information is summarized below:

- Analysis and Communication Aids/System Properties: All object types have unique characteristics or properties that must be specified. For example the SYNONYMS, or ATTRIBUTES, ASSERT, KEYWORDS, TRACE-KEY, SEE-MEMO, DESCRIPTION statements fall into this category.
- System Input/Output Flow: This refers to the interaction between the target system and its environment. The object types that are involved include INPUTS, OUTPUTS, INTERFACES, and SETS.
- System Structure: Both information structures and processing structures are included. This is based on relationships between objects such as SUBPARTS ARE, PART OF, COMPOSED OF, COMPONENT IN, SUBSET OF, SUBSETS ARE, DETERMINES, and A FUNCTION OF. The involved object types are CONDITION, ENTITY, EVENT, INPUT, INTERFACE, OUTPUT, PROCESS, PROCESSOR, RELATION, SET and SYSTEM_PARAMETER.
- Data Structure: It refers to the relationships which exist among data used and/or manipulated by the system as seen by the users. The object types that are involved include ATTRIBUTE, ELEMENT, ENTITY, GROUP, INPUT, OUTPUT, RELATION and SET.
- Data Derivation: This refers to the way in which data are

manipulated or derived by the target system. The involved object types are CONDITION, ELEMENT, ENTITY, GROUP, INPUT, INTERFACE, OUTPUT, PROCESS, PROCESSOR, RELATION, and SET.

- **System Control and Dynamics:** This aspect specifies the target system's behavior over time. The involved object types are CONDITION, ENTITY, EVENT, INPUT, OUTPUT, PROCESS, PROCESSOR, and SET.
- **Project Management:** This refers to the documentaon of the design project identification of people involved and their responsibilities. These are specified by using statements such as RESPONSIBLE_PROBLEM_DEFINER, SECURITY, and SOURCE. All object types are involved.

3.2 Reporting Facilities

In this section an overview of the reporting facilities available in both POD and PSA are reviewed. This material is presented first by examining the reporting facilities basically common to both POD and PSA and then by reviewing the reporting facilities specific to each tool.

3.2.1 Common Facilities

Numerous reporting facilities exist in POD and PSA that present essentially overlapping information about the design of the of the target system. The major aspects of commonality are software structure, resource consumption, and cross reference.

Both POD and PSA produce reports on the software structure of the target system. These reports are useful for understanding the logical flow of the software design. In POD, the TREE command turns on this report generating mechanism. The name of a job or a module is specified as the "root" of the structure, then all the modules invoked directly or indirectly by the root are presented. The report has an indented depth-first format. "Depth-first" implies that CALL statements are expanded as they occur. An example follows:

```

TREE      UPDATE_RECORD

UPDATE_RECORD
  SWAP_IN
  WRITE_RECORD
  DISPLAY
    ACCESS_RECORD
  CLEANUP
    WRITE_RECORD
  SWAP_OUT

```

In PSA, the STRUCTURE report and the UTILIZES ANALYSIS report may be used to present the software structure. In the STRUCTURE report, the input names are assigned to the top level. All the names related directly to the top level names are designated level two names, the names related directly to level two names will be level three names, and so on. All the different relations specified between names in two consecutive levels are also indicated as shown in the following example:

1	UPDATE_RECORD	PROCESS	
2	SWAP_IN	"	(UTILIZED)
2	WRITE_RECORD	"	(")
2	DISPLAY	"	(")
3	ACCESS_RECORD	"	(")
2	CLEANUP	"	(")
3	WRITE_RECORD	"	(")
2	SWAP_OUT	"	(")

The UTILIZES ANALYSIS report represent the interactions among processes defined by the use of SUBPARTS ARE and UTILIZES statements. An example is given below:

1	1.0	UPDATE_RECORD
2	1.1	SWAP_IN
3	1.2	WRITE_RECORD
4	1.3	DISPLAY
5	1.3.1	ACCESS_RECORD
6	1.4	CLEANUP
7	1.4.1	WRITE_RECORD
8	1.5	SWAP_OUT

The reports on resource consumption are provided by POD and PSA. Those produced by POD includes MODREP (Module Report), DEVREP (Device

Report), FILREP (File Report), and MEMREP (Memory Report). Those produced by PSA are the RESOURCE CONSUMPTION ANALYSIS report and the FREQUENCY report. In POD, the MODREP command generates a report of the total time spent at the specified device by each module of the specified job. The frequency at which each module is executed by the job is also reported. Conventionally, all times are in milliseconds. This report is illustrated as below:

MODREP	UPDATE_RECORD		CENTRAL_PROCESSOR		
MODULE	LOCAL	PERCENT	TOTAL	PERCENT	TIMES EX.
UPDATE_RECORD	42.0	19.5%	215.5	100%	1.0
SWAP_IN	23.3	10.8%	23.3	10.8%	1.0
WRITE_RECORD	78.0	36.2%	78.0	36.2%	3.0
DISPLAY	14.0	6.5%	46.0	21.3%	1.0
ACCESS_RECORD	32.0	14.8%	32.0	14.8%	2.0
CLEANUP	3.0	1.4%	39.0	18.1%	1.0
SWAP_OUT	23.3	10.8%	23.3	10.8%	1.0

The POD DEVREP report presents the total time consumed at each device by the specified job as depicted below:

DEVREP	UPDATE_RECORD	
1	215.5	CENTRAL_PROCESSOR
2	103.0	DISK_A
3	96.3	disk_B

The POD FILREP report presents the total time spent accessing each file on the specified device by the specified job as shown below:

FILEREP	UPDATE_RECORD	DISK_A
FILE	USAGE	PERCENT
R2D2	79.0	81.4%
C3P0	18.0	18.6%

And finally, the POD MEMREP report provides an overlay memory layout of jobs in the system. It portrays each module's size and describes its role in the overlay structure specified for the job. For example:

MEMREP

MODULE	LOCAL MEMORY REQUIREMENTS(WORDS)	RESIDENT
--------	----------------------------------	----------

UPDATE_RECORD	6000	YES
READ_INDEX	900	YES
READ_MSG	820	NO
SEND_MSG	350	NO
WRITE_INDEX	650	NO

OVERLAY	AREA	TYPE	SIZE	CONTENT
S_1		SEGMENT	14%	WRITE_INDEX, READ_MSG
R_1		REGION	14%	SEND_MSG, S_1

The RESOURCE CONSUMPTION ANALYSIS report of PSA presents processors' consumption of resources in performing the specified process and its component processes. The analyzer performs a walk of the SUBPARTS_UTILIZES network which starts at the specified PROCESS name and extends toward to the specified number of levels. The resource consumption and frequency information is computed for each visit to each process in the structure. Since a process may be visited many times, its resource consumption may be computed many times as well. This report is similar to the POD DEVREP except that POD incorporates the workload information to compute the system performance and PSA does not. Depending upon the viewpoints, two formats - BYPROCESSOR and BYRESOURCE - may be used in PSA to provide resource consumption status about each PROCESS. This is illustrated by the following example:

ROOT PROCESS: UPDATE_RECORD

PROCESSOR CENTRAL_PROCESSOR IS INVOKED 600 TIMES PER HOUR

RESOURCE CONSUMED	AMOUNT CONSUMED	MEASURED IN
CPU_TIME	215.5	MSEC

PROCESSOR DISK_A IS INVOKED 800 TIMES PER HOUR

RESOURCE CONSUMED	AMOUNT CONSUMED	MEASURED IN
DISK_A_TIME	103.0	MSEC

The above report is presented in BYPROCESSOR format. If BYRESOURCE format is used, the report may appear as below:

POOT PROCESS: UPDATE_RECORD

RESOURCE CPU-TIME MEASURED IN MSEC

PROCESSOR CONSUMING	FREQUENCY	AMOUNT CONSUMED
CENTRAL_PROCESSOR	800/HOUR	215.5

RESOURCE DISK_A TIME MEASURED IN MSEC

PROCESSOR CONSUMING	FREQUENCY	AMOUNT CONSUMED
DISK_A	800/HOUR	103.0

The FREQUENCY report of PSA presents frequency information about all INPUTS, OUTPUTS, PROCESSES, and EVENTS defined with respect to the HAPPENS relations. It may be used together with the RESOURCE CONSUMPTION ANALYSIS report to analyze the system performance.

POD and PSA both provide the reporting facilities to determine the dependencies of the target system on various objects. The POD command XREF produces a cross-reference list. Two parameters are specified. The first parameter can specify a JOB or ALL - all modules in the Module Specification Section. The second parameter can specify MODULES, DEVICES, FILES or PARAMETERS. For example:

XREF ALL DEVICES

STORE: DISK_A, CENTRAL_PROCESSOR, MEMORY_1
FLASH: DISK_B, CENTRAL_PROCESSOR, MEMORY_1
FORWARD: DISK_A, CENTRAL_PROCESSOR, MEMORY_1

The PSA DYNAMIC_INTERACTION report presents all objects dynamically related to the given object such as INPUTS, EVENTS, CONDITIONS, and PROCESSES. It is shown in the form of a matrix, the relationship between two objects is also indicated by a specific entry at the interaction of a particular row and column which represents the two objects respectively.

(i,j)	Meaning
A	Row i TRIGGERS Column j
B	Row i BECOMING-TRUE IS CALLED Column j
N	Row i BECOMING-FALSE IS CALLED Column j
C	Row i INTERRUPT-CAUSES Column j
P	Row i TERMINATION-CAUSES Column j
Q	Row i UTILIZES Column j
X	Row i is the same as Column j

Column Names

1	hourly-emp-processing-init	EVENT
2	salaries-data-collection	EVENT
3	salaries-paycheck-production	EVENT
4	hourly-data-collection	EVENT
5	new-employee-processing-init	EVENT
6	hourly-paycheck-production	EVENT
7	termination-processing-init	EVENT
8	salaries-employee-processing	PROCESS
9	salaries-paycheck-validation	PROCESS
10	salaries-exp-update	PROCESS

Row Names

1	salaries-employee-processing-init	EVENT
2	salaries-paycheck-production	EVENT
3	salaries-data-collection	EVENT
4	hourly-data-collection	EVENT
5	hourly-paycheck-production	EVENT
6	new-employee-processing-init	EVENT
7	termination-processing-init	EVENT
8	hourly-data-collection	EVENT
9	salaries-employee-processing	PROCESS
10	salaries-paycheck-production	PROCESS

	1	2	3	4	5	6	7	8	9	10
1	:				:			A		:
2	:			X	:					:
3	:		X		:				A	:
4	:	X			:					:
5	:				:	X				:
6	:				X					:
7	:				:		X			:
8	:			X	:					:
9	:	O	O		:			X	Q	Q
10	:				:					:

Figure 3-4: PSC Dynamic Interaction Matrix

An example is shown in Figure 3-4.

3.2.2 POD Specific Facilities

Due to the unique POD performance calculation mechanisms, there are numerous facilities available within POD that are not available within PSA. Some of these are summarized below.

- The MODUTIL command reports which portion of a workload's device utilization is attributable to each module.

- The FILUTIL command reports which portion of a workload's device utilization is attributable to each file.
- The MODSYS command reports which portion of the total utilization of a device is attributable to each module.
- The FILSYS command reports which portion of the total utilization of a device is attributable to each file.

The POD Performance Calculator component analyzes the system through the POD System Description File. At the conclusion of this analysis, the user is placed in an interactive environment in which further more detailed performance reports can be requested. Some of these are listed below.

- Principle Results Report: For each workload, the report indicates the average response time per transaction, the average throughput in transaction per hour, and the CPU utilization associated with that workload. Total CPU utilization is also reported.
- CPU and I/O Utilization Report: The total utilization of each server (device) is presented in the first column, followed by a "per workload" breakdown of that utilization.
- Response Time Profile Report: This report gives the response time, in milliseconds, at each server (device) in the system.
- Average Queue Length Report: This report indicates the average queue length at each server (device). The first column contains the total for each server and the remaining columns provide a "per workload" breakdown.
- Average Number Waiting Report: This report presents the average number of requests actually waiting for processing at each server.
- Waiting Time Profile Report: The time spent waiting - in milliseconds - at each server is presented.
- Service Rate Degradation Report: This report presents the ratio of total response time to actual service time for each workload at each server.
- Server Residency Profile Report: This report presents the percent of time that a transaction spends waiting for or receiving service at each server in the system.
- Queue Length Distribution Report: For a server, this report indicates the aggregate distribution for all workloads, as well as an individual distribution for each workload.

- **Concurrency Report:** This report presents the percent of time that two or more servers within the system are simultaneously active.
- **Memory Report:** This report presents information related to the utilization of main memory in the system that is being evaluated. The information consists of: the average and maximum multiprogramming levels for each domain. The average number of transactions waiting to be loaded into each domain, the percent of time that a domain is full and one or more transactions are waiting to be loaded into that domain, the distribution of the number of transactions waiting to be loaded into each domain.

Another facility unique to POD is the Feasibility Testing Facility. The purpose of this facility is to automate parametric studies of performance and to interface to a graphics facility to produce performance results in a graphic manner. This provides the design analysts with the ability to examine the sensitivity of various performance parameters in a dynamic manner rather than simply through numeral/tabular format. An example analysis from this facility is shown in Figure 3-5.

3.2.3 PSL/PSA Specific Facilities

PSA reports present information on various aspects of the target system such as Analysis and Communication Aids, System Input-Output Flow, System Structure, Data Structure, Data Deviation, System Size and Volume, Control and Dynamics, and Project Management. These aspects may be essential to logical system design, but POD, currently designed to facilitate addressing performance issues of system design, does not provide the reporting facilities for some of these aspects. Data Structure, Data Deviation, and System Input-Output Flow are the most apparent ones. Moreover, the efforts that PSL/PSA makes in the areas of Communication and Analysis Aids as well as Project Management are not emphasized in POD.

- **Data Structure:** Reports for the data structure aspect include CONTENTS, CONTENTS ANALYSIS, CONTENTS COMPARISON, EXTENDED PICTURE IDENTIFIER ANALYSIS, RELATION STRUCTURE, SECURITY ANALYSIS and SUBSET ANALYSIS.
- **Data Deviation:** Reports for the data deviation aspect include DATA ACTIVITY INTERACTION, ELEMENT PROCESS ANALYSIS, ELEMENT PROCESS USAGE, EXTENDED PICTURE, FUNCTION_FLOW_DATA_DIAGRAM, RELATION STRUCTURE, and PROCESS SUMMARY.

GRAPH 1

12-Mar-79 20:52
STORE_WKL RESPONSE_TIME

```

READ 40
VARY EXFACT
FROM .8 TO 2 BY .2
STOPIF STORE_WKL
RT > 8.0
RUN
GRAPH 1

```

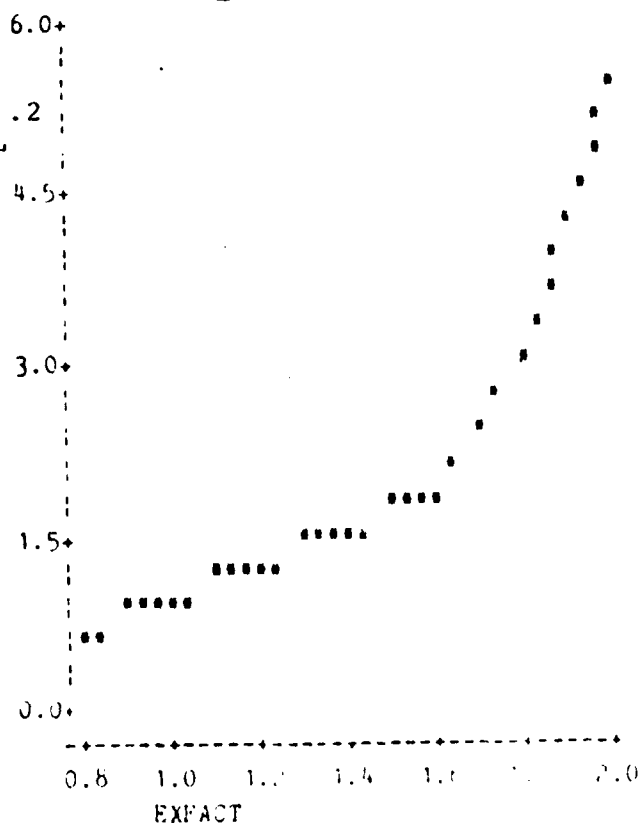


Figure 3-5: Sample FTS Interaction

- System Input-Output Flow: Reports for this aspect include EXTENDED PICTURE, PICTURE, and DATA ACTIVITY INTERACTION.
- Communication and Analysis Aids: Reports for this aspect include ASSERTION CONSISTENCY ATTRIBUTES, DICTIONARY, KEYWORD IN CONTEXT, LAYOUT, and NAME LIST.
- Project Management: Reports for this aspect include DATA BASE SUMMARY, and LIST CHANGES.
- Control and Dynamics: Reports for the control and dynamics aspect include PROCESS CHAIN, and UTILIZES ANALYSIS.
- System Structure: Reports for the system structure aspect include EXTENDED PICTURE, and PICTURE.

- In addition, some reports are for all aspects. They are: STRUCTURE, NAME SELECTION, PUNCH COMMENT ENTRIES, and FORMATTED PROBLEM STATEMENT.

The various PSA reports can also be classified on the basis of the purpose which they serve:

- Data Base Modification Reports: They constitute a record of changes for error correction and recovery.
- Reference Reports: They present information in the data base in various formats.
- Summary Reports: They present collections of information in summary form, or gathered from several different relationships.
- Analysis Reports: They provide various types of analysis of the information in the data base.

3.3 Functional Processing

In the previous two sections approaches of POD and PSL/PSA were compared as to their input specifications and their reporting facilities. In this section their approaches to functional processing are examined. Both tools provide basically similar facilities for tabulation and simple computation on the information specified via their respective system description syntax. They are however, significantly different in the approaches which are adopted to analyze the system performance.

3.3.1 PSL/PSA Approach

PSL/PSA provides various kinds of facilities for the analysis of information processing systems. These facilities may be divided into static analysis and dynamic analysis. Static analysis deals with the structure of the information processing system. Dynamic analysis deals with the system behavior over time.

Because of the fact that PSL only provides limited facilities for describing the dynamic aspects of the information processing system, the System Description Language (SDL) is intended as a replacement for PSL in

situations where simulation is necessary. SDL is an ongoing research project at the University of Michigan that is being designed to facilitate upward compatibility with PSL and, in addition, to provide a simulation capability. The object types and relations for the static aspects have been retained in SDL essentially in the form they appear in PSL. The other aspects (system size and volume, system dynamics) have been replaced by three new aspects - system modeling, resource management, and system dynamics and control. The System Definition Manager (SDM) provides all the facilities for static analysis and non-quantitative dynamic analysis available in PSA. In addition, the SIMSCRIPT MODEL GENERATOR is incorporated to produce a SIMSCRIPT language program from the SDL description of a information processing system contained in a System Description Data Base. This approach, illustrated in Figure 3-6, allows for the automatic generation of simulation programs concurrent with the design of other information processing system aspects.

A SIMSCRIPT simulation program consists of five ordered sections. the PREAMBLE, the MAIN program, EVENT routines, PROCESS routines, and miscellaneous routines. The MODEL GENERATOR consists of modules to generate the first four parts of the SIMSCRIPT program and a few of the miscellaneous routines. The user-supplied routines not generated by the MODEL GENERATOR usually perform such functions as reporting the simulation results, evaluating common repetitive expressions, and dumping the state of the model of an unmanageable internal condition is detected during the simulation.

3.3.2 POD Approach

POD, like PSL/PSA, also provides both static analysis and dynamic analysis facilities. The Static Analysis commands invoke mechanisms that summarize the information in the System Description File (SDF) or information that can be computed from the SDF by interpreting the module statements. The Dynamic Analysis commands invoke mechanisms that extract the performance information from the SDF, create a model of the complete system, and perform computation on the model to predict the performance of the target system.

The Performance Calculator component of POD is the mechanism invoked for Dynamic Analysis. It uses the workload information and the device usage for each job to create a multiclass queueing network model, a simple example of which is depicted in Figure 3-7.

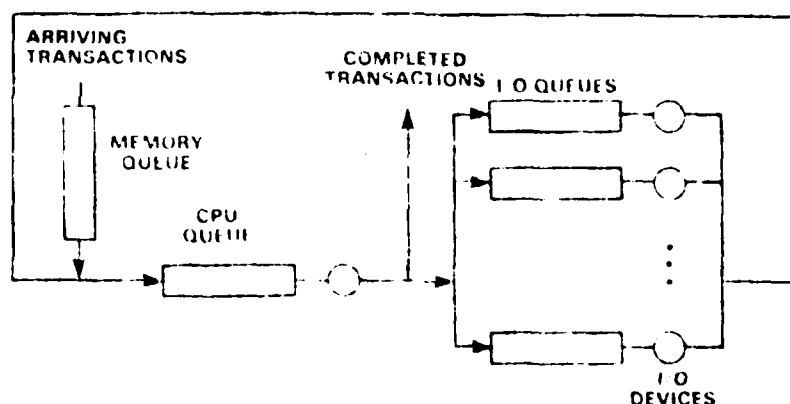


Figure 3-7: Basic POD/PC Model

The rectangles in the figure indicate the locations of queues and the circles represent servers such as CPUs or I/O devices. An arriving transaction may at first have to wait in the memory queue until space for its loading and execution becomes available.

The transaction then receives a burst of CPU processing which terminates when an I/O request is generated. The I/O request might, for example, represent a swap, or perhaps an initiation of a data base record transfer. At this point, the transaction proceeds to the appropriate I/O device, an I/O transfer is initiated. Upon completing this transfer, the transaction returns to the CPU queue. The alternating cycle of CPU bursts and I/O transfers continues until the processing requirements of the

transaction are satisfied. The transaction then terminates and leaves the system via the "COMPLETED TRANSACTION" arrow.

The generality of the model enables the analyst to represent cases where several transactions are in main memory and active at the same time. One transaction may be using the CPU while another is using an I/O device. Similarly, several I/O devices may be providing service to several different transactions at the same time.

The model is parameterized using a service time array for each workload. Each component service time in this array is the total time that a single average job in a workload spends in receiving service from a system resource. Each location (server) in this array represents the pertinent system devices such as CPUs, I/O devices, channels and others². Quantitative results on the performance of these devices (e.g., utilization), and on the performance of workloads consuming these resources (e.g., response time and throughput) are available. The method of solution is based on the theory of multiclass queueing networks supplemented by sophisticated engineering approximation techniques.

Much has been written about the comparative advantages and disadvantages of the simulation (SDL/SDM) and analytic queueing based (POD) approaches. The simulation approach provides a modeling mechanism that can be detailed and hence is powerful. The analytic approach, in contrast, is tractable for a much smaller class of problems. On the other hand, simulation is computationally expensive often making a performance analysis of any practical size costly and time consuming. Analytic solution techniques, based on efficient numerical methods are, in contrast, rapid and accurate for the class of problems they solve thereby providing a cost-effective alternative to simulation.

²For example, POD has recently been extended to include such tactical devices as radar screens, gun mounts and others.

4 INTERFACE FEASIBILITY AND REQUIREMENTS

In this chapter we address the feasibility of and the requirements for interfacing POD with PSL/PSA. The chapter assesses PSL/PSA in terms of its ability to describe required POD system inputs. In addition, consideration is given to possible PSL/PSA extensions to facilitate the interface process.

There seem to be several possible approaches for interfacing POD and PSL/PSA. These are

- Manual translation of PSL/PSA input to POD input
- Automatic (machine driven) translation of PSL/PSA input to POD input.
- Translation of PSL/PSA output reports into POD input.

In each case the object of the translation process is to translate a PSL/PSA system specification into a POD System Description File. This file could then be utilized by the performance calculation component of POD for further performance specification and analysis. Any of the above three approaches could be utilized. Of course, the creation of an automated procedure for the translation process is the most desirable alternative, since in the long run this approach tends to minimize the level of effort involved in the translation. Moreover, for the most part, it should be possible to extend the reporting facilities of PSL/PSA to automatically produce, as output, a POD System Description File.

4.1 Direct Translation

A translation program to convert a PSL system specification into a POD system description file needs to be concerned with the following tasks:

- Mapping of PSL/PSA constructs to POD constructs
- Identifying sources of data redundancy, inconsistency and/or

incompleteness in a PSL/PSA specification which, in turn, would without resolution lead to an incomplete POD System Description File

- Resolving the above mentioned ambiguities in the translation process through manual or automated intervention.

Mapping of the PSL/PSA constructs into the POD System Description Syntax is the primary task of the PSL/PSA to POD translation program. Some mappings such as converting the PSL <process alpha> UTILIZES <process beta> syntax into POD MODULE alpha CALL beta syntax are straightforward to implement. Others such as those involving conditional invocation or TRIGGERing of processes may be more complex. This is because the probability of invocation may need to be specified and such information is not currently standard PSL/PSA input. Rather it exists in PSL/PSA currently as natural language English comments embedded in the PSL/PSA specification, if it exists at all. In such cases the translator will need to "flag" such occurrences and provide for the user to "fill in" the missing information manually.

Handling of synonyms and combining references to different fields within records to, for example, determine total file utilization requires some bookkeeping. This bookkeeping can be facilitated by designing a symbol table that allows many-to-one mappings. More extensive bookkeeping may be needed to keep track of redundant specifications and cross references (such as the joint use of the TRIGGERS and TRIGGERED BY constructs). Similarly, determining and flagging places where a specification is incomplete or inconsistent will require careful analysis and maintenance of internal state information. Also it should be noted that both PSL/PSA and POD are evolving systems whose syntaxes are likely to be extended and otherwise modified. Any translation tool would have to take this into account and be maintainable in such a manner that changes to the PSL and POD syntaxes could be incorporated into the translator in a modular manner. This implies that a grammar or table driven translator would be the appropriate implementation approach.

A summary of the main PSL constructs that contain information useful for building a POD model is given in the following subsections and the accompanying table. These are organized according to the main sections of

a POD system description file.

4.1.1 Configuration Specification

The POD Configuration Specification Section provides the user with the facility to describe the computer environment. Specifically, the Configuration Specification Section provides facilities for specifying individual hardware devices (e.g., CPU, channels, disks) and, in the case of I/O devices, pertinent information concerning the files resident on the devices.

Hardware resource consumption descriptions are not given as much attention in PSL as the logical architecture of a system. However, the RESOURCE and PROCESS sections both contain useful information that can be picked up in the translation. The RESOURCE section is roughly analogous to a POD device type description. A PSL processor roughly corresponds to a POD device, but can also contain what POD would consider device type information. Attributes and parameters specifying operations and device speeds can be specified for RESOURCES. The specific rate at which a RESOURCE is CONSUMED (in POD terms the rate at which an operation is performed) is specified in the PROCESSOR section. Typically, hardware configuration information in PSL would be expressed in syntax of the form:

```
DEFINE RESOURCE <resource-name>;
.
.
.
CONSUMED BY <processor-name>
AT RATE OF <number> PER <attribute>;
.
.
.
MEASURED BY <unit>;
```

This could be translated to a POD configuration specification of the form:

```
DEVICE <resource-name>
.
.
.
RATE = <number>
```

For example, a PSL description of the form:

```
DEFINE RESOURCE central-processor;
    CONSUMED BY processor1
    AT RATE OF 2 PER instruction;
    MEASURED BY microseconds;
```

could be translated to the POD hardware configuration specification:

```
DEVICE central-processor
TYPE = cpu
RATE = .5 &MIP (MILLION instructions/sec)
```

Translation of disk device (and more unconventional device) descriptions and the mapping of files to individual disk or drum devices may be more complex. This is because the choice of attribute names and meanings are currently user defined in PSL. Whether and how a given device operation is described and the manner of describing device connectivity/topology is a choice left to the PSL user.

4.1.2 Workload Specification

The POD Workload Specification Section provides the user with the facility to describe the frequency with which programs or tasks are executed in a computerized environment. In this context, the term workload refers to a stream of jobs (requests for processing) arriving at the computer system from some external source. For example, a stream of position updates generated by a navigation subsystem might constitute one workload, and a stream of range/bearing values from a radar subsystem might constitute another.

The POD Workload Specification Section provides facilities for specifying the characteristics of the workload and characterizing the memory regions allocated for execution of the workloads' job stream. Workload descriptors within the POD Workload Specification Section are used to specify relevant workload characteristics while domain descriptors are used to characterize a workload's memory domain.

PSL uses the HAPPENS construct as its main way of describing the

frequency of events in an application environment. Either a PROCESS invocation or an INPUT can HAPPEN with a certain frequency per time interval. A conversion program will need to identify those processes and inputs that represent the top level of a function (transaction) invocation; i.e. those inputs and events that represent interactions between a computerized process and the external world. These top level interactions correspond to POD jobs. The jobs can then, optionally, be combined into workloads for modeling and reporting purposes. Since there are currently no primitives in PSL for specifying workload (as opposed to job in the POD sense) characteristics, this last step will probably have to be done manually after the translation process.

The top level interfaces to PSL processes need to be distinguished from internal interfaces between processes (modules) within the system. The latter correspond to subroutine calls or interprocess communication. In addition, INPUT occurrences and PROCESS invocations need to be correlated in order to avoid counting what is really a single process invocation multiple times. This could happen, for example, when both an input occurrence frequency and a process invocation frequency are specified for the same event.

The translation process will have to convert a PSL statement of the form:

```
[ INPUT | PROCESS ] HAPPENS [ <system-parameter> |
    <integer> | <real-number> ]
    TIMES-PER <interval>
```

to a POD statement of the form:

```
WORKLOAD <name> TYPE=<category>
.
.
.
JOB-STREAM = <top-level-process-name>
RATE = [ <system-parameter> | <integer> | <real-number> ]
    / <interval>
END
```

For example, the PSL statements:

```
DEFINE PROCESS transaction-type-1;
.
.
.
HAPPENS 20 TIMES-PER second;
.
.
.
ATTRIBUTES ARE workload-type transaction, mpl 2;
```

would become the POD sequence:

```

WORKLOAD transaction_type_1 TYPE = transaction
ARRIVAL_RATE = 20 & /second
MPL = 2
JOB_STREAM = transaction_type_1
END

```

It is important to note that the concepts of workload, multiplicity and domain are not currently a standardized part of PSL. These are important aspects of a POD input specification. They could, however, be incorporated into PSL either as new primitives or by defining standardized attribute types.

4.1.3 Software Specification

The POD Module Specification Section contains the specifications of all modules (software components) for each job in the system. This section provides the user the facility to specify pertinent control flow and resource consumption information. Modules may be specified in any order since each is "compiled" individually by POD. The module specifications for separate jobs are not physically segregated - they are all specified in a single Module Specification Section. Individual modules of a job are related to a job's root module which in turn relates to a workload.

POD describes the characteristics of the software it models in terms of two main types of specification, describing respectively:

- the interrelationships between modules
- the resource consumption of individual modules.

These correspond to the PSL UTILIZES and CONSUMES statements respectively. In addition, PSL describes file (data set /entity) access through the use of REFERENCES, ADDS, MODIFIES, REMOVES, CREATES, and DESTROYS primitives. These correspond to reads and writes to a file in a POD module specification.

It may not always be clear in a PSL description whether an access is a write or a read/write, and whether the number of physical reads (or writes) is really the same as the logical number specified in the PSL REFERENCES

(ADDS, MODIFIES, etc.) statement. In addition, certain algorithmic type information such as how many times a loop is executed or the relative probability that a module will take one of two alternative paths at a branch point are not generally included in a PSL description.

Module linkage and data (file) access is described as part of the PSL PROCESS section. Resource consumption is described in the PROCESSOR and RESOURCE sections. A PERFORMS primitive links the PROCESSOR to a specific PROCESS (module). Attributes can be used to associate specific resource usage information with this linkage.

The primitives to be converted are as follows. The PSL statements:

```
DEFINE PROCESS <name1>;
. . .
UTILIZES <name2> [, <name3>];
```

are converted to the POD statements:

```
MODULE <name1>
. . .
CALL <name2>
CALL <name3>
```

For example,

```
DEFINE PROCESS alpha;
. . .
UTILIZES beta, gamma;
```

would be converted to:

```
MODULE alpha
. . .
CALL beta
CALL gamma
. . .
END
```

Data set access in PSL is converted to file reads and writes in POD as follows. The PSL:

```
PROCESS <process-name> REFERENCES [<system_parameter> ;
<number>] <entity_name> IN <data_set_name>
```

is converted to

```
MODULE <process_name>
EST <data_set_name> USAGE = [<system_parameter> ; <number>]
READS
```

Similarly, the PSL syntax

```
[ADDS | MODIFIES | REMOVES] [<system_parameter> | <number>]
<entity_name>] [IN | FROM] <data_set_name>
```

is translated to:

```
EST <data_set_name> USAGE = [<sysparm> | <number>] WRITES
```

For example, the PSL syntax:

```
PROCESS alpha
.
.
REFERENCES 26 data_type1_records IN file1;
MODIFIES 12 data_type2_records IN file2;
```

would be translated to

```
MODULE alpha
.
.
EST file1 USAGE = 26 READS
EST file2 USAGE = 12 WRITES
```

Note that the translator would normally have to assume that there is exactly one physical read or write per logical file access or provide for a user override that supplies additional I/O information.

Resource consumption specification translation is somewhat more complex since it involves tracing and mapping through multiple PSL statements grouped in two or three PSL specification sections: the PROCFSOR (task) specification, the PROCESS (module) specification, and potentially the RESOURCE (device type) specification. A PSL resource usage description would take the form:

```
DEFINE PROCESS <process-name>;
.
.
ASSERT <attribute-name> IS <attribute-value>;
.
.
DEFINE PROCESSOR <processor-name>;
.
.
CONSUMES <resource-name> AT RATE OF [<system-parameter> |
<number>] PER <attribute-name>;
.
.
PERFORMS <process-name>;
```

This would map into the POD statements:

```
MODULE <process-name>
EST <resource-name> USAGE = [<system-parameter> | <number>] *
<attribute-value> & <units>
```

For example, the PSL statements

```

DEFINE PROCESS data-tabulation;
ASSERT no_data_record IS 200;
.
.
.
DEFINE PROCESSOR task1;
CONSUMES central_processor AT RATE OF 60 /* instructions */
PER no_data_record;
PERFORMS data-tabulation;

```

would be translated into

```

MODULE data_tabulation
EST central_processor USAGE = 12000 INSTRUCTIONS

```

Here 12000 = 200 (the number of data records) times 60 (the number of instructions per data record). Note that the variety of ways that attributes can be defined and used in PSL can make the translation of syntax involving these constructs fairly complex. /* . . . */ indicates a comment in PSL.

Linkage between modules can also be specified indirectly through an event in PSL using the INCEPTION-CAUSES and TERMINATION-CAUSES primitives to link a PROCESS (module) to an EVENT and the TRIGGERED BY primitive to link a second process to the same event. EVENTS themselves can be linked to processes using the ON INCEPTION OF, ON TERMINATION OF, and TRIGGERS primitives. In addition, events can be linked to each other using the CAUSES and CAUSED BY primitives and be asserted to occur periodically using the HAPPENS primitive. Analyzing and interpreting these indirect connections may complicate the POD model building process.

4.1.4 Defining System Parameters

The POD Global Declaration Section provides a section for global definitions of user defined unit keywords and performance parameters.

In most cases PSL system parameter definitions can be mapped into POD global parameters without problem. Thus, the PSL syntax

```

DEFINE SYSTEM-PARAMETER <name>;
.
.
.
VALUE IS <value>;

```

can be mapped into

```

GLOBAL DECLARATION
  . . .
  DEFINE <name> = <value>
    or
  PARAMETER <name> = <value>
END

```

For example,

```

DEFINE SYSTEM-PARAMETER several;
VALUE IS 4;
DEFINE SYSTEM-PARAMETER number_of_terminals;
VALUE IS 200;

```

maps into

```

GLOBAL DECLARATION
DEFINE several = 4
PARAMETER number_of_terminals = 200
END

```

4.2 Potential Extensions to PSL to Facilitate Performance Specification

Translation to POD system specification files and performance modeling of systems based on PSL specifications would be facilitated if the PSL syntax for expressing performance related quantities was formalized to a greater extent than is currently done. Key performance related information in PSL is currently specified using three basic constructs:

- system parameters
- attributes
- procedure, description and comments constructs written in free format English.

Information related to branching conditions and probabilities (where multiple paths can be taken through a procedure) is often expressed as free format comments in a PSL PROCEDURE or DESCRIPTION statement. Information related to the relative frequency of taking alternate paths need not be specified in PSL at all. In addition, information related to the mapping of files (PSL entities and sets) to mass storage devices and specification of device types with multiple and/or complex operations would be easier to handle with standardized syntax.

Probably the simplest way to extend PSL so as to provide a more comprehensive interface to POD would be to select certain standard parameter and attribute names to be used to specify performance. If this were to be done the following characteristics would be likely candidates for standardization:

- program size
- path length (average number of instructions)
- explicit statement of logical record length and block size for files
- device attributes for standard devices such as disks
- device connectivity where networks or channels are involved
- some notion of workload, multi-programming level, memory sharing, and domains
- the mapping of disk files (sets) onto specific physical devices (This might also be specifiable as a PSL relationship.)
- physical record size (block factor) for files (sets and entities)
- process or job priorities

Selection of specific attributes and parameters for standardization would need to be done in collaboration with the ISDOS group and users. A side benefit would be that the semantics (meaning) of attributes and parameters (as well as the choice of attribute and parameter names) would then be the same for all PSL/PSA users. This would significantly increase the accuracy and reliability of resulting models.

4.3 Interactive Tools to Aid the Translation Process

Even if parameters and attributes related to performance are standardized in PSL, there are likely to be gaps in the PSL descriptions of system performance. This can occur either because performance related quantities were not known at the time of initial system specification or because performance information was not considered at that time. Because of this, it may be worthwhile to develop an interactive translation

facility that processes a PSL description file and then prompts the user for missing information or permits the user to resolve ambiguities in the system description. Such a tool might also be useful as an aid to users building a POD system description file from scratch. In either case it would need to be menu driven and need to maintain information about system characteristics in a semantically interpretable internal state. Libraries of device type descriptions and other information could be maintained to speed model building.

SPECIFICATION

POD APPROACH

PSL APPROACH

Device Descriptions

Part of the Configuration Specification Section.

Defined in the RESOURCE and PROCESSOR sections of a PSL description.

Device Attributes

Specific attributes are defined for most common devices. For example, an instruction execution RATE is defined for CPUs. A DEVICE MAP, a data transfer RATE, SEEK, REVOLUTION TIME, CAPACITY, and existence of RPS is defined for disks. I/O path and specialized device characteristics may also be specified. In addition, the concepts of device class and multiplicity are available to allow large number of devices to be specified more concisely.

Device attributes are user defined. There are currently no standard conventions for attribute names, but there is active discussion about introducing such conventions. There is no direct equivalent of device class. Multiplicity and device connectivity could be expressed using attributes, but there is no evidence that this is being done currently in PSL descriptions.

Device Topology

ATTACH statement

Could be specified as a user defined attribute.

File Description Information

Specified as part of the file catalog of the Configuration Specification Section. Information specified includes FILE_SIZE, RECORDS_SIZE, BLOCK_FACTOR.

Usually specified as part of a SET description, but can also be specified in an ENTITY, INPUT or OUTPUT description.

File characteristics are usually specified indirectly by specifying the entity, input or output composition in terms of elements and groups of elements and then giving the size of the elements. Could also be specified by user defined attributes.

• SPECIFICATION

Workload Descriptions

Workload Type

Workload Arrival Process

Job stream

Dispatching Priority

Multiprogramming Level

POD APPROACH

Specified in the Workload Specification Section.

- Five types are defined: Interactive, Transaction, Periodic, Cyclic and One-shot.

A workload arrival process is described by parameters describing arrival rates, number of user terminals and user think-time.

A probability distribution of job streams is allowed for each workload. Each job stream consists of a sequence of module/software invocations.

Workload Priority

MPL, domains

PSL APPROACH

Specified indirectly/implicitly using the HAPPENS statement.

Can be specified as a user defined attribute.

Processes, inputs and events are specified to HAPPEN with a given frequency.

The top level process (main program) or an external input could be used to define a job. This would generally include a HAPPENS statement that specifies arrival rate.

No concept of combining processes (jobs) into workloads exists, but this could be added as a user defined attribute.

Could be specified using user defined attributes.

Could be specified using user defined attributes.

WORKLOAD SPECIFICATIONS

SPECIFICATION	POD APPROACH	PSL APPROACH
Specification of Software Characteristics	Module Specification Section	PROCESS section
Subroutine Calls	Call statement	UTILIZES statement
CPU Activity	EST <processor name> USAGE IS <instructions> This is converted to a CPU service time using information in the configuration specification section.	A PROCESS is linked to a RESOURCE via a PROCESSOR, which performs a PROCESS and CONSUMES a RESOURCE. The RESOURCE is consumed at a rate corresponding to the speed of the processor. A user-defined attribute can be used to specify the number of operations required to perform the process.
I/O Activity	Specified as number of physical reads and writes to the file. Translation to I/O service time is dependent upon file characteristics specified in the file catalog entry of the Configuration Specification and the performance characteristics of the device upon which the file is mapped.	Various types of access to entities, inputs and outputs can be specified with a numerical parameter that can be used to specify number of records accessed. A certain number of INPUTS can be RECEIVED; OUTPUTS are GENERATED. ENTITIES can be ADDED, REFERENCED, MODIFIED, REMOVED, CREATED or DESTROYED. RELATIONS can be CREATED or DESTROYED. A system parameter can be used instead of a number to specify number of accesses.
Algorithmic Specification	LOOP and TEST CASE statements with associated probabilities are provided by POD.	This level of detail is not normally addressed by PSL.
Program Terminations	TERMINATES	TERMINATED BY and TERMINATION CAUSES constructs are provided.

SOFTWARE SPECIFICATIONS

SPECIFICATION

POD APPROACH

PSL APPROACH

User Defined Units

Specified in Global Declaration Section

Specified anywhere in PSL description. Relatively straightforward to translate.

Performance Parameters

Specified in Global Declaration Section

System parameters Relatively straightforward to translate.

Probability Distributions

Can be specified anywhere in a System Description File.

Usually specified in a descriptive form using natural English as part of a DESCRIPTION or PROCEDURE primitive. The specification is interpreted as a character string and not given further semantic meaning by PSA.

Memory Allocation

The POD OVERLAY-MAP is used to depict memory overlays of individual jobs.

There is no direct equivalent in PSL.

OTHER SPECIFICATIONS

5 SUMMARY AND CONCLUSIONS

This report has addressed the feasibility of and requirements for interfacing two powerful software engineering tools - POD (Performance Oriented Design) and PSL/PSA (Problem Statement Language/Problem Statement Analyzer). POD is a software engineering tool for the life cycle management of system performance. It is a tool which provides a structured machine readable format for representing a system's hardware architecture, software structure and external load demand. Moreover, based on this representation, POD provides the vehicle to calculate system performance values such as response time, throughput and device utilization. PSL/PSA is a computer-aided structured documentation and analysis technique used for analysis and documentation of requirements and preparation of functional specifications. The capability to describe systems in computer processable form is based on the system description language PSL. The ability to record the description in a data base, incrementally modify it, and perform analysis is based on the problem statement analyzer PSA.

In considering the feasibility of and requirements for interfacing POD with PSL/PSA, PSL/PSA was assessed in terms of its ability to describe the required POD input specifications. In addition, consideration was given to possible PSL/PSA extensions to support POD. The major finding of this report are summarized below.

- The major objective of both POD and PSL/PSA is the reduction of system life cycle costs through the provisions of automated software engineering techniques to enhance the system development process. Some overlap in functionality does exist between these two tools; however, for the most part each tool has its own distinct direction. The major focus of PSL/PSA is requirements and functional specification; the major focus of POD is design and performance analysis. Creating an interface between these tools can therefore be expected to provide a more complete and comprehensive software engineering environment than is provided currently by either tool alone.
- Though the major focus of PSL/PSA is requirements and functional specification, research work is currently in progress at the University of Michigan to transform PSL/PSA so that it can interface to SIMSCRIPT simulation programs. Such an interface

may have the potential to provide performance related information. However, specification of most performance inputs to PSL/PSA is not standardized at this time. Moreover, simulation is generally computationally expensive in comparison to analytic techniques such as those embedded in the POD performance calculator. An interface between the two tools would permit the PSL/PSA analyst to have access to comprehensive and efficient POD performance calculation routines directly.

- The two basic approaches to the PSL/PSA - POD interface are manual translation and machine-driven translation. In either case the object of the translation process is to translate PSL/PSA system specification input and/or output into POD system description input format. The machine-driven translation process is the more desirable alternative, since in the long run this approach will minimize the level of translation effort. Information required for the translation is readily available for the most part and, is summarized in detail in Chapter 4 of this report.
- The requirements for such an automated translation program include:
 - Mapping of PSL/PSA constructs to POD constructs.
 - Identifying sources of data redundancy, inconsistency and/or incompleteness in a PSL/PSA specification which, in turn, would without resolution lead to an incomplete POD System Description File.
 - Resolving the above mentioned ambiguities in the translation process through manual or automated intervention.
- In order to facilitate the translation/interface procedure it is desirable that PSL/PSA be extended to include standardized performance input specifications. Currently much performance information in PSL/PSA is optional and is specified only in natural language comment form or via user defined attributes whose performance semantics are not fully amenable to machine analysis. Standardized performance specifications in PSL/PSA will tend to reduce the amount of manual and/or automatic intervention required to complete the translation process and act to facilitate the interface to POD.

ATE
LME